Using Inductive Reasoning to Find the Cause of Process Delays

Evgeniy Vasilyev
Tokyo Institute of Technology
Tokyo, Japan
Email: vasilyev.e.aa@m.titech.ac.jp

Diogo R. Ferreira
IST – Technical University of Lisbon
Lisbon, Portugal
Email: diogo.ferreira@ist.utl.pt

Junichi Iijima
Tokyo Institute of Technology
Tokyo, Japan
Email: iijima.j.aa@m.titech.ac.jp

***Abstract*—Delays in business processes can have negative consequences for organizations, such as extra costs, missed deadlines, poor service, etc. Although they are easy to detect, it may be hard to find the actual reason for such delays. In this work we develop an approach to find the cause of delays based on the information recorded in an event log. The approach is based on a logic representation of the event log and on the application of decision tree induction to separate process instances according to their duration. In this decision tree, the delayed instances immediately stand out, and by following the path in the tree it is possible to extract a rule that characterizes those instances and therefore provides a possible explanation for the delay. We illustrate the approach in a set of experiments with event logs that are generated by simulation of a purchase process. In each experiment, there is a different cause of delay, and the approach succeeds in finding that cause. Additional experiments show that the approach is scalable and tolerant to noise.**

I. Introduction

In process mining, an event log is usually seen as a sequence of events which can be analyzed from different perspectives. For example, there are several algorithms that can be used to extract a control-flow model from the sequence of tasks recorded in an event log (e.g., the *α-algorithm* [1], the *heuristics miner* [2], the *genetic miner* [3]). There are also techniques to analyze the interaction between process participants by extracting a social network from the event log [4], [5]. And there are ways to calculate performance indicators and to detect bottlenecks based on the timestamp of events [6], [7]. A practical case study that includes these different perspectives (i.e. the control-flow, the organizational, and the performance perspective) can be found in [8].

In this work we are looking at the performance perspective, and in particular at the problem of finding the reason why some process instances become delayed. With delayed, we mean that a process instance takes longer than usual to complete. This could be due to many reasons, such as, for example: when an activity is assigned to a certain user in particular; when two activities that should be performed by the same user are performed by different users; when a customer orders an item that is not currently in warehouse; etc.

In general, the reasons for process delay may be related to the control-flow perspective, to the organizational perspective, to the data flow or the specific case data associated with a process instance. In the typical event logs that are used for process mining, one can find information about the control-flow perspective and about the organizational perspective, while information about the data flow is not usually available. Nevertheless, previous works [9]–[11] focused on data flow as the primary cause for process delays, and therefore the proposed methods require either the use of external data or the availability of additional attributes in the event log. One of the reasons for this was a certain inability of previous approaches based on propositional logic to take full advantage of the knowledge that is usually embedded in the event log.

In this work we address this problem with an approach based on first-order logic for data representation. Typically, each event in an event log records the process instance (i.e. the case id), the task that has been executed, the user who performed the task, and the completion timestamp of the event. We capture this knowledge as a set of logic predicates, and we use *inductive logic programming* [12] – specifically, a decision tree learner known as TILDE [13] – to classify process instances into different groups according to their duration. For practical purposes, we define the duration of a process instance as the difference between the timestamps of the last and the first event recorded for that instance. The result of applying such classification is a decision tree where each path from the root to a leaf provides a rule that characterizes a certain group of instances. This rule can be interpreted as a reason why that group of instances has a certain duration.

The structure of the paper is as follows. Section II highlights the difference between the present approach and related works. In Section III we introduce a process that will serve as a running example throughout the paper. Section IV defines the predicates that are used to represent the event log. Section V discusses the induction of decision trees in general, and of logic decision trees in particular. Section VI describes a set of experiments to illustrate the approach and to cover the main problems that may arise in practice. Section VII discusses the results and reports on additional experiments concerning scalability and noise-tolerance. Finally, section VIII concludes the paper.

II. Related work

Inductive logic programming (ILP) has been used before in the area of process mining, but not for the purpose we describe here. For example, [14] uses ILP and in particular TILDE to learn the preconditions of each activity from the event log and from a set of negative events that are generated artificially to discover the process model. Ref. [15] uses a related technique called *inductive constraint logic* [16] to learn declarative models from an event log containing traces labeled as compliant or non-compliant. Also, [17] uses ILP combined with a planning algorithm in a framework that supports the continuous learning of business processes.

Some recent works [9]–[11] that are more closely related to our approach use propositional classification techniques for the analysis of the performance perspective. In particular, [11] proposes an architecture for a business process intelligence tool suite which makes use of classification techniques for the analysis of process behavior. Ref. [10] analyzes how case data affects the routing of process instances with the help of decision trees. Ref. [9] describes an approach that uses decision trees to find out whether there is a relationship between the workload of resources and the fact that the process gets overdue (i.e. the case duration exceeds a threshold); for that purpose, the authors make use of an event log that is enriched with workload information.

Our present approach differs from [9]–[11] in several ways:

- We use the most traditional type of event log, with the set of attributes that are typically available for process mining: namely, the case id, the task, the performer, and the completion timestamp of the event. All the information that is necessary for analysis is derived directly from the event log and we do not use any external data source.
- We use first-order logic and in particular a set of predicates expressed in a PROLOG-like syntax, which allows us to represent knowledge in an expressive and convenient way for induction. For example, `flow/3` or `flow(I,A,B)` is a predicate with three arguments to assert the fact that in process instance `I` (where `I` is the case id), activity `A` is followed by activity `B`. The event log, described in terms of the first-order predicates `event/4`, `followed/7`, and `duration/2`, can be used as direct input to the classifier.
- In our approach we use decision trees to classify cases (i.e. process instances) in terms of their actual duration as recorded in the event log, rather than on predefined and discrete categories such as "late" and "on time" as in [9], [11]. This allows us to group cases according to duration and to study the cause for delayed instances. In practice, it is often hard to define the notion of delay beforehand, i.e. to say whether a particular instance is delayed without comparing it to other cases. By separating cases into different groups, our approach is able to identify the cases that take longer to complete, without the need for a precise definition of delay (e.g., as a threshold).

III. AN EXAMPLE PROCESS

Figure 1 illustrates a purchasing process that will be used in our experiments. This process can be described as follows. First, an employee fills out a requisition form (a) and sends it for approval by a supervisor (b). If it is not approved, the requisition is archived (c). If it is approved, the product is ordered from a supplier (d) and two branches will run in parallel: at the warehouse an employee receives the product (e) and updates the stock (f); and at the accounting department someone else will take care of payment (g). When these two branches complete, the requisition is closed (h).

Each activity in this process is assigned to some user, and the user is selected at run-time from a set of users who are authorized to perform the task. Figure 1 provides a visual indication, by means of horizontal bars, of the probability of a user being selected to perform a certain task. For example, both $u4$ and $u5$ may perform task c, with $u4$ having a higher probability of being assigned than $u5$.

Another aspect that is depicted graphically in Figure 1 is the duration of each task. For simulation purposes, the duration of each task is assumed to follow a normal distribution with some mean μ and standard deviation σ, with the additional requirement that it must be non-negative (because duration cannot be negative). This is illustrated by having a bell shape associated with each activity. The distance from the origin and the width of this bell shape provide an idea of the mean and standard deviation of the corresponding probability density function associated with duration.

These process parameters – i.e. the probability of a user being selected for a certain task, and the mean and standard deviation for the duration of each activity – are quantified in Tables I and II.

TABLE I. TASK ASSIGNMENT PROBABILITIES

	$u1$	$u2$	$u3$	$u4$	$u5$	$u6$	$u7$	$u8$	$u9$
a	1.0	–	–	–	–	–	–	–	–
b	–	0.7	0.3	–	–	–	–	–	–
c	–	–	–	0.6	0.4	–	–	–	–
d	–	–	–	–	–	0.4	0.6	–	–
e	–	–	–	–	–	–	–	0.5	0.5
f	–	–	–	–	–	–	–	0.5	0.5
g	–	–	–	–	–	0.4	0.6	–	–
h	–	–	–	0.6	0.4	–	–	–	–

TABLE II. DURATION OF ACTIVITIES (IN HOURS)

	μ	σ
a	24.0	12.0
b	48.0	24.0
c	24.0	12.0
d	24.0	12.0
e	96.0	48.0
f	24.0	12.0
g	96.0	48.0
h	48.0	24.0

Using these parameters, we wrote a program to simulate the process in Figure 1 and to generate event logs for our experiments. Because the assignment policy and task duration have a probabilistic nature, there is some level of randomness in the generated event logs. An excerpt of a generated event log is shown in Table III. This shows the first three cases in the event log. In these three instances, it is possible to recognize several features of the process. For example, case 1 has a trace in the form $abdgefh$, while case 2 has a trace in the form abc (we defined a 75% probability of approval); also, in case 1 activity g happens before e and f, while in case 3 it happens afterwards. Furthermore, it is possible to see the user assignment policy in action. For example, activity f is performed by user $u9$ in case 1 and by user $u8$ in case 3. The values in the last column of Table III are not part of the original event log, but they are calculated from the timestamp of events, as explained in the next section.

IV. EVENT LOG REPRESENTATION

As described in Section I, we address the analysis of process delays as a classification problem. In particular, we

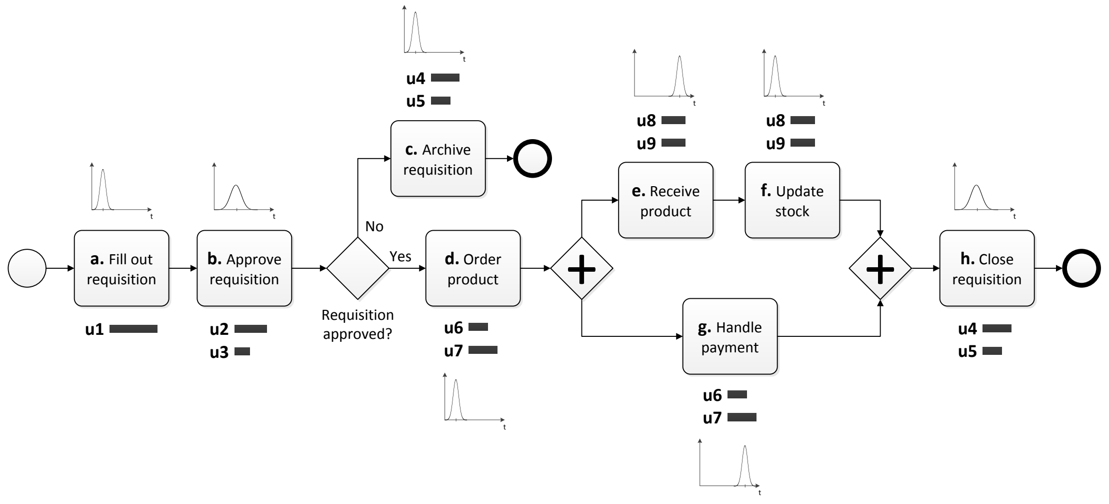

Fig. 1. A purchase process

TABLE III. EVENT LOG

case id	task	user	timestamp	$\mathbf{h}(\cdot)$ (see Def. 4)
1	a	u1	2012-11-25 14:19:09	0.000
1	b	u2	2012-11-28 00:03:46	57.744
1	d	u6	2012-11-28 01:45:22	59.437
1	g	u7	2012-12-01 21:59:47	151.677
1	e	u8	2012-12-02 03:30:24	157.187
1	f	u9	2012-12-02 22:20:37	176.024
2	a	u1	2012-12-03 10:27:27	0.000
1	h	u4	2012-12-04 07:39:36	209.341
2	b	u3	2012-12-04 08:36:42	22.154
2	c	u5	2012-12-05 23:04:46	60.622
3	a	u1	2012-12-12 01:13:46	0.000
3	b	u2	2012-12-14 23:20:06	70.106
3	d	u6	2012-12-15 21:14:01	92.004
3	e	u8	2012-12-19 16:57:46	183.733
3	f	u8	2012-12-19 19:17:04	186.055
3	g	u6	2012-12-20 02:06:03	192.871
3	h	u4	2012-12-21 00:15:38	215.031
...	...	...	...	...

use ILP, which is a machine learning technique that combines first order logic for data representation together with inductive reasoning for learning hypotheses from data. The induction process is discussed in more detail in the next section, while in this section we will concentrate mainly on data representation.

In ILP, hypotheses are derived (i.e. induced) from two sources: the first is the *knowledge base*, which contains a set of observed examples described in terms of logic facts, and the second is the *background knowledge*, which is a set of domain-specific concepts and rules that apply to all examples and that are defined in terms of logic clauses. The knowledge base and the background knowledge define the language bias for induction. The derived hypotheses will be expressed in terms of such language bias (i.e. facts and rules).

In our case, the information in Table III, which represents a typical event log (the first four columns), is to be captured as a set of logic facts (i.e predicates). These facts concern the tasks that were performed in each process instance, the user who performed each task, and also the duration of the process instance. Such set of facts will represent the knowledge base. On the other hand, there is set of rules to define some common concepts, such as handover of work between users; these will be added as background knowledge.

In order to define these facts and rules in a precise way, we introduce the following definitions and predicates:

Definition 1 (Event)**.** *An event is a tuple $\epsilon = (i, a, u, t)$ where: i is the case id that identifies the process instance; a is the name of the activity (or "task") whose completion generated the event; u is the name of the user who performed the task; and t is the timestamp (i.e. date and time) when the event was recorded. We define $\mathbf{i}(\epsilon) = i$ and $\mathbf{t}(\epsilon) = t$.*

Definition 2 (Trace)**.** *A trace is a sequence of events $\tau = \langle \epsilon_1, \epsilon_2, ..., \epsilon_n \rangle$ such that $\mathbf{i}(\epsilon_1) = \mathbf{i}(\epsilon_2) = ... = \mathbf{i}(\epsilon_n)$ and $\mathbf{t}(\epsilon_1) \leq \mathbf{t}(\epsilon_2) \leq ... \leq \mathbf{t}(\epsilon_n)$. In other words, the events belong to the same process instance and are ordered by timestamp. We define $\mathbf{i}(\tau) = \mathbf{i}(\epsilon_1) = ... = \mathbf{i}(\epsilon_n)$.*

Definition 3 (Event log)**.** *An event log is a set of traces $\mathcal{L} = \{\tau_1, \tau_2, ..., \tau_N\}$ where $\mathbf{i}(\tau_j) \neq \mathbf{i}(\tau_k), \forall_{\tau_j, \tau_k \in \mathcal{L}}$.*

Definition 4 (Time and duration)**.** *For a given trace $\tau = \langle \epsilon_1, \epsilon_2, ..., \epsilon_n \rangle \in \mathcal{L}$ we define the time of an event $\epsilon_k \in \tau$ as $\mathbf{h}(\epsilon_k) = \mathbf{t}(\epsilon_k) - \mathbf{t}(\epsilon_1)$. In particular, $\mathbf{h}(\epsilon_1) = 0$. We also define the duration $\mathbf{d}(\tau)$ of a trace $\tau = \langle \epsilon_1, \epsilon_2, ..., \epsilon_n \rangle$ as the time of the last event: $\mathbf{d}(\tau) = \mathbf{h}(\epsilon_n) = \mathbf{t}(\epsilon_n) - \mathbf{t}(\epsilon_1)$. In practical scenarios, we will measure $\mathbf{h}(\cdot)$ and $\mathbf{d}(\cdot)$ in hours.*

Definition 5 (Knowledge base)**.** *A knowledge base is a set of statements that represent the facts in an event log $\mathcal{L}$ in terms of the predicates `event/4`, `followed/7`, and `duration/2`.[1] In particular, the knowledge base for a given event log $\mathcal{L}$ comprises the following kinds of statements:*

- `event`$(i, a, u, \mathbf{h}(\epsilon))$. $\forall_{\tau \in \mathcal{L}}, \forall_{\epsilon=(i,a,u,t) \in \tau}$
- `followed`$(i, a', u', \mathbf{h}(\epsilon_k), a'', u'', \mathbf{h}(\epsilon_{k+1}))$. $\forall_{\tau \in \mathcal{L}}, \forall_{\epsilon_k=(i,a',u',t'), \epsilon_{k+1}=(i,a'',u'',t'') \in \tau}$
- `duration`$(\mathbf{i}(\tau), \mathbf{d}(\tau))$. $\forall_{\tau \in \mathcal{L}}$

[1]We use the PROLOG notation `predicate/arity` to indicate the number of arguments for each predicate.

Def. 5 means that an event log $\mathcal{L}$ will be translated into a set of facts using the predicates `event/4`, `followed/7`, and `duration/2`. For example, the logical representation of case 1 in Table III will be:

```
event(1, a, u1, 0.000).
event(1, b, u2, 57.744).
event(1, d, u6, 59.437).
event(1, g, u7, 151.677).
event(1, e, u8, 157.187).
event(1, f, u9, 176.024).
event(1, h, u4, 209.341).
followed(1, a, u1, 0.000, b, u2, 57.744).
followed(1, b, u2, 57.744, d, u6, 59.437).
followed(1, d, u6, 59.437, g, u7, 151.677).
followed(1, g, u7, 151.677, e, u8, 157.187).
followed(1, e, u8, 157.187, f, u9, 176.024).
followed(1, f, u9, 176.024, h, u4, 209.341).
duration(1, 209.340815).
```

From this knowledge base, it is possible to derive additional facts about the tasks and users in this process, and about how these tasks and users follow each other. The following definition provides the rules to derive additional facts:

Definition 6 (Background knowledge). *We define background knowledge as a set of general rules that allow deriving additional facts about the behavior observed in an event log. In particular, we define the following predicates:*[2]

- *`task(I,A) :- event(I,A,_,_).`*
  (True if task `A` appears in case `I`.)
- *`user(I,X) :- event(I,_,X,_).`*
  (True if user `X` participates in case `I`.)
- *`performer(I,A,X) :- event(I,A,X,_).`*
  (True if user `X` performs task `A` in case `I`.)
- *`flow(I,A,B) :- followed(I,A,_,_,B,_,_).`*
  (True if there is a flow from task `A` to task `B` in case `I`.)
- *`handover(I,X,Y) :- followed(I,_,X,_,_,Y,_).`*
  (True if there is handover of work from user `X` to user `Y` in case `I`.)
- *`together(I,X,Y) :- event(I,_,X,_), event(I,_,Y,_).`*
  (True if users `X` and `Y` work together in case `I`.)

These new predicates concern the control-flow perspective (`task/2` and `flow/3`) as well as the organizational perspective (`user/2`, `performer/3`, `handover/3` and `together/3`). In particular, the predicates `handover/3` and `together/3` allow to determine, respectively, if there is handover of work between users and if users work in joint cases, as defined in [4]. It would be possible to define more predicates, for example to detect if two activities are performed in parallel or if there is a causal relationship between them, as in the *α-algorithm* [1]. However, the rules above will be enough to begin with in the experiments of Section VI.

[2]Again, we use a PROLOG notation to define these predicates.

V. INDUCTION

In this work the induced hypotheses are represented as decision trees, which are one of the most popular tools in machine learning and data mining. Given a set of objects with a collection of attributes, a decision tree divides the objects into a set of mutually exclusive classes according to the values of their attributes. The distinctive feature of decision trees is that the conditions that determine class membership are structured in the form of a tree. At the root of the tree we check the value of a certain attribute and, depending on that value, we follow one of the available branches; at the following node, there will be a new decision based on a different attribute; and so on, until we reach a leaf of the tree. The leaf is labeled with the name of the class that the object belongs to.

The induction of decision trees from relational data (i.e. objects with a collection of attributes) can be done using popular algorithms such as ID3 [18] and C4.5 [19]. Both ID3 and C4.5 are able to generate decision trees where each node is associated with a certain attribute and each branch coming out of that node is associated with a different value for that attribute. The induction of a decision tree is therefore a learning process, where one learns the attribute values that are associated with the objects belonging to each class. Such process is called propositional learning or attribute-value learning.

Decision trees can also be used for concept learning by associating a predicate with each node, as shown in Figure 2. In this case, there are two branches coming out of each node in order to represent the two possible outcomes for such predicate (true or false). This kind of tree is referred to as *logical decision tree* and its induction can be seen as a form of ILP. A popular algorithm for inducing logical decision trees is TILDE, which is an upgrade of C4.5 with logical predicates.[3]

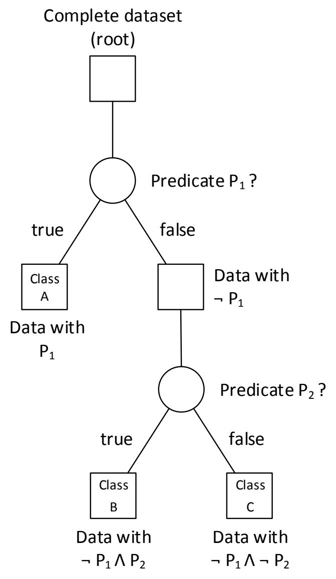

Fig. 2. Structure of a logical decision tree

When decision trees are used for classification, the output variable is discrete (i.e. the object belongs to one from a

[3]TILDE is developed by K. U. Leuven and it is part of the ACE data mining system that can be found at: *https://dtai.cs.kuleuven.be/ACE/*

set of classes). However, in some applications the output variable is continuous (i.e. each class corresponds to a range of values). The induction of such tree usually involves linear regression and therefore this kind of tree is sometimes referred to as a *regression tree* [20]. This is precisely the case in our present problem, where process instances are to be classified according to their duration, which is a continuous variable. TILDE supports the induction of both logical decision trees and logical regression trees, so it becomes an especially useful tool for our purpose.

VI. EXPERIMENTS

In this section we report on a set of experiments using event logs obtained through simulation of the business process described in Section III. Each event log contains $N = 100$ process instances (i.e. cases). In each experiment, the simulator inserts deliberate delays in the process instances, according to a given condition. The event log is then translated into a knowledge base according to Def. 5. This knowledge base, together with the background knowledge of Def. 6, is provided as input to TILDE, which generates the decision tree. Our goal is to check whether the decision tree is able to capture the correct cause for the delay. These experiments were chosen to illustrate some problems that may arise in practice.

Experiment 1: When user u8 participates in the process, the process gets delayed, on average, about 80 hours.

In this experiment, we inserted a delay with mean $\mu = 80.0$ and standard deviation $\sigma = 15.0$ in those process instances that have some activity performed by user $u8$. The delay is inserted in the total duration of the process instance, as indicated by predicate `duration/2`. After the event log has been converted into a knowledge based, running TILDE on this knowledge base together with the background knowledge of Def. 6 yields the following result:

```
duration(I,X) [287.75] 100
task(I,d) ?
+--yes: [324.88] 85
|       user(I,u8) ?
|       +--yes: [355.30] 59
|       |       together(I,u6,u9) ?
|       |       +--yes: [372.95] 32 [8.97]
|       |       +--no:  [334.38] 27 [10.12]
|       +--no:  [255.84] 26 [11.02]
+--no:  [77.38] 15 [7.28]
```

The first line contains the target predicate and the two values following it are the average duration and the total number of cases, respectively. Each predicate with a question mark represents a node, and the labels `yes` and `no` represent the branches (whether the predicate is true or false). For each branch, the first number in square brackets indicates the average duration in the group, the second is the number of examples that can be found in the group, and the third number, again in square brackets, and when applicable, provides the standard deviation within the group.

The tree begins by dividing the cases into two groups, depending on whether the case contains the task d or not. The cases that do not contain this task have an average duration of about 77 hours, which is significantly shorter than the average duration for all cases (about 288 hours). Going back to the process model in Figure 1, one realizes that this corresponds to the process instances where the requisition was not approved and therefore the task d was not performed. This group was discovered in all experiments, so we will not refer to it again in subsequent experiments.

The remaining cases are further divided into two subgroups. The first subgroup contains the cases where both task d was executed and user $u8$ performed some task. In this tree, the predicate `user(I,u8)` is a child node with respect to the root node with predicate `task(I,d)`. Therefore, the rule for this child node can be expressed as a conjunction of the predicates `user(I,u8)` and `task(I,d)`. The duration of cases in this subgroup is about 355 hours on average, which is around 100 hours above the average of the opposite subgroup. The rule and the duration of the first subgroup correspond to the conditions that were set for this experiment, and therefore we can say that TILDE found the cause of the delay.

Since we use regression, TILDE tries to discover rules for all distinct groups of process instances with similar duration. An event log may give origin to a large number of such groups. However, not all of them may be of interest to the process analyst. In order to define whether a particular group is important or not, the process analyst can calculate the difference between the target variable in the branches created by a given predicate, and compare it to the same difference calculated for other predicates.

For example, in this experiment we can find a further subdivision of the tree with predicate `together(I,u6,u9)` that forms two groups with a duration difference of 38.57 hours. Comparing this to the same difference for the predicate `user(I,u8)` which is 99.46 hours, and for the predicate `task(I,d)` which is 247.50 hours, the analyst may conclude that the division formed by the predicate `together(I,u6,u9)` is not as relevant as the previous ones. However, it may be the case that a difference of 30 hours is important and therefore those groups should be considered. The fact that the predicate `together(I,u6,u9)` creates additional groups in this experiment is due to the randomness in the event logs, which was mentioned in Section III, and which may generate unintended patterns.

It is possible to adjust TILDE's sensitivity by changing a parameter that sets the minimal number of instances to be included in each leaf. By doing so, the analyst may characterize the group of delayed instances more precisely. However, it may also happen that some meaningful groups are lost. For example, if we change the value of this parameter from 10 to 30 instances and run TILDE over the same event log, the node that is associated with predicate `task(I,d)` is lost:

```
duration(I,X) [257.86] 100
user(I,u8) ?
+--yes: [341.40] 59 [5.91]
+--no:  [137.65] 41 [15.24]
```

Experiment 2: When users u3 and u4 work together in the process, there is a delay, on average, of about 100 hours.

In this experiment, TILDE yields the following tree:

```
duration(I,X) [233.35] 100
```

```
task(I,d) ?
+--yes: [266.61] 80
|       together(I,u3,u4) ?
|       +--yes: [359.49] 12 [13.08]
|       +--no:  [250.22] 68 [6.36]
+--no:  [100.31] 20
        together(I,u3,u4) ?
        +--yes: [162.45] 6 [9.77]
        +--no:  [73.67] 14 [5.66]
```

The cause for delay can be found in the nodes with the predicate `together(I,u3,u4)`.

Experiment 3: When different users perform tasks d and g the process gets delayed, on average, about 70 hours.

In this experiment, we simulate the scenario where the tasks d and g are performed by different users. These tasks should be performed by the same user and, whenever it does not happen that way, the process suffers some delay. TILDE produces:

```
duration(I,X) [264.16] 100
task(I,c) ?
+--yes: [77.11] 11 [7.25]
+--no:  [287.27] 89
        together(I,u6,u7) ?
        +--yes: [314.78] 45 [8.36]
        +--no:  [259.14] 44 [7.66]
```

Since there is no special predicate either in the knowledge base or in the background knowledge for expressing the fact that two tasks are performed by different users, TILDE derives this rule in terms of others predicates, particularly with the help of predicate `together(I,u6,u7)`. If we go back to Figure 1 to analyze the assignment policies, then the fact that $u6$ and $u7$ work together implies that tasks d and g are indeed performed by different users. It should be mentioned that the knowledge about assignment policy is not hard to get from the event log if it is unknown, what is more important is that there is no guarantee that the initially defined background knowledge will contain all necessary predicates for describing the reason for process delay.

Such kind of problem will inevitably arise in practice. In order to mitigate it, the analyst should be ready to modify the background knowledge during the course of analysis based on some additional insights about the business process that can be obtained from studying the output decision trees. In general, running TILDE over the same event log with different sets of predicates may help the analyst discover the true reason for the delay. For example, when analyzing the tree above, the analyst may come up with idea that the real reason for process delay is in fact that tasks d and g were performed by different users and, in order to verify this idea, the following predicate could be included in the background knowledge of Def. 6:

- `sameperformer(I,A,B) :- event(I,A,X,_), event(I,B,X,_).` (True if tasks `A` and `B` are performed by the same user in case `I`.)

Then TILDE derives the following tree from the same event log, which can be interpreted without prior knowledge of the assignment policy, and which captures the cause for delay more explicitly:

```
duration(I,X) [264.16] 100
task(I,c) ?
+--yes: [77.11] 11 [7.25]
+--no:  [287.27] 89
        sameperformer(I,d,g) ?
        +--yes: [259.14] 44 [7.66]
        +--no:  [314.78] 45 [8.36]
```

In addition, a condition can be equivalently expressed in terms of different predicates, and TILDE may not choose the easiest expression for interpretation. An example of equivalent expressions for the same condition can be seen in the groups of process instances where task c is performed. In Experiments 1 and 2 this fact is expressed with the predicate `not(task(I,d))`, while in this experiment it is expressed with predicate `task(I,c)`.

Experiment 4: When u6 hands over work to u7 then the process completes earlier, on average about 80 hours.

Since the process completes earlier, here we are testing for a "negative delay", and TILDE yields:

```
duration(I,X) [192.29] 100
task(I,d) ?
+--yes: [240.79] 72
|       performer(I,d,u6) ?
|       +--yes: [212.04] 31 [11.25]
|       +--no:  [262.53] 41 [8]
+--no:  [67.55] 28 [5.85]
```

According to the definition of `handover(I,X,Y)` (see Def. 6), this predicate is true when user `X` performs a task which is followed by another task performed by user `Y`. Users $u6$ and $u7$ execute tasks d and g, and d is followed by g in the process model. However, in the event log it is possible to find the sequences $dgef$, $degf$ and $defg$, since e and f are in parallel with g. In the sequences $degf$ and $defg$, even if task d is executed by $u6$ and g is executed by $u7$, the predicate `handover(I, u6, u7)` will be false. Therefore, the tree above does not capture the true reason for the delay.

This problem could be addressed by defining appropriate predicates to capture the fact that some activities are being performed in parallel. However, in order to demonstrate how this approach can be used together with process mining algorithms, we address the problem in a different way. Let us assume that we run some process discovery algorithm (such as the α-algorithm) over the event log, and from that algorithm we obtain the following relationships between activities:

```
flow(a,b).
flow(b,c).
flow(b,d).
flow(d,e).
flow(e,f).
flow(d,g).
flow(f,h).
flow(g,h).
```

These facts capture the sequence flow (i.e. the arrows) between the activities in the model of Figure 1. We can add these facts to the knowledge base and remove the predicate `flow/3` rule from the background knowledge. We can also redefine the predicate `handover/3` to take into account the actual flow between activities:

- `handover(I,X,Y) :- event(I,A,X,_), event(I,B,Y,_), flow(A,B).`
  (True if there is handover of work from user `X` to user `Y` in two activities that are connected by a flow.)

After this update, the newly induced tree correctly describes the reason for the (negative) delay:

```
duration(I,X) [192.29] 100
task(I,d) ?
+--yes: [240.79] 72
|       handover(I,u6,u7) ?
|       +--yes: [182.04] 18 [13.15]
|       +--no:  [260.38] 54 [6.79]
+--no:  [67.55] 28 [5.88]
```

We have conducted additional experiments with more complicated conditions, e.g., *"when u3 hands over work to u6 and user u8 executes task f then the process gets delayed"*, and again the tree captured the correct cause for the delay.

VII. Discussion

From the experiments above and other experiments that we carried out, it is possible to conclude the following:

1) It is possible to find the cause of delays in process instances, if indeed the cause is related to facts that can be derived from the event log. These facts should be represented by appropriate predicates.
2) In general, any additional knowledge (e.g., about the process model, or about the assignment policy) that can be extracted from the event log can be helpful in determining and interpreting the cause of delays. This additional knowledge may be obtained, for example, through the use of process mining algorithms.
3) The predicates defined in this paper should not be considered as being universally valid. This approach assumes that there will be a process analyst who understands the business process and can adjust the background knowledge according to domain-specific needs. Moreover, in some cases it may be necessary to change the initially defined background knowledge during the course of analysis as the analyst gets more insights into the business process, as illustrated in Experiments 3 and 4. Also, it may happen that not all of the defined predicates will be used in the tree (e.g., predicate `flow/3` in Def. 6).
4) The cause for delay can be equivalently expressed in different ways with the predicates available in the language bias (e.g., `not(task(I,d))` and `task(I,c)`). Some forms may be easier to interpret than others.

Besides the experiments described in the previous section, we carried out additional experiments to assess the approach with respect to scalability and noise-tolerance.

A. Scalability

Regarding scalability, we carried out a series of tests using the same scenarios as in Section VI. We increased the number of process instances N successively by amounts of 10, from $N = 100$ to $N = 1000$. It would have been possible to go further, but the range $100 \leq N \leq 1000$ was enough to produce an observable trend. This trend was similar for all experiments. Figure 3 illustrates the results for Experiment 2.

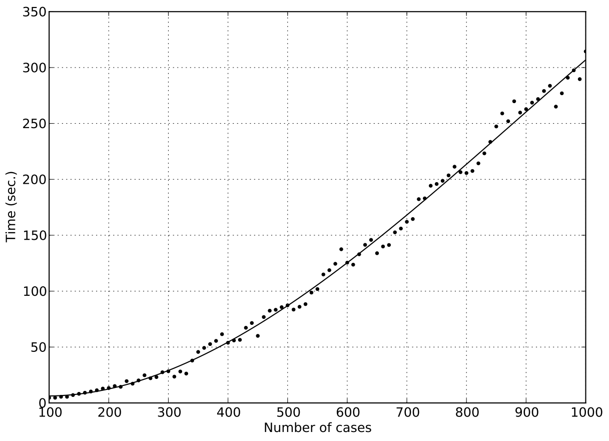

Fig. 3. Total run time for TILDE with increasing number of cases.

For $N = 100$, TILDE takes about 5 seconds to induce the decision tree. This includes the time to start the program and to load the knowledge base and the background knowledge, which is fairly quick. Most of the time is spent in induction. For $N = 1000$, TILDE takes about 314 seconds to complete (i.e. over 5 minutes). However, at this point the trend is practically linear, as illustrated in Figure 3. The solid line in the plot is a least-squares fit with a cubic function $f(x) = ax^3 + bx^2 + cx + d$ where: $a = -2.50 \times 10^{-7}$, $b = 0.000664$, $c = -0.119$, and $d = 11.7$. At $x = 1000$, the slope of this function is $f'(1000) = \frac{df}{dt}\big|_{1000} \simeq 0.46$ seconds/case, i.e. an additional 4.6 seconds per each increase of 10 cases.

The fact that tree induction appears to scale linearly agrees with the claim that, under some general assumptions, the complexity of TILDE is linear with the number of examples [21].

B. Noise tolerance

With regard to noise, we carried out two different experiments. In the first experiment we changed the amount of delay that was inserted in the event log and the number of process instances that were affected by it. The results suggest that, in order to be able to discover the group of delayed instances, the delay should be more than one standard deviation of the process duration (i.e. if the process duration has mean μ and standard deviation σ, then the duration of a delayed instance should be at least $\mu + \sigma$). The standard deviation for the duration of the process described in Section III is about 60 hours, so this should be regarded as the minimum amount of delay to be used in experiments. On the other hand, the number of process instances that suffer from delay should be statistically significant. For example, if there are very few delayed cases, say 2 or 3 out of 100, and the delay is not much more than one standard deviation, it will be difficult to isolate this group. In this case, the delayed instances are likely to be bundled together with other instances in some other group that satisfies a rule that is not the cause for the delay.

In the second experiment we inserted in the event log the same types of noise as used in [22]–[24], namely:

(a) missing head, i.e. the removal of events at the beginning of a case;
(b) missing body, i.e. the removal of events in the middle of a case;
(c) missing tail, i.e. the removal of events at the end of a case;
(d) swap tasks, i.e. the interchange of two events in a case;
(e) remove task, i.e. the removal of an arbitrary event from a case;
(f) mix all, i.e. a combination of all of the above.

After trying out each type of noise separately, we used mostly option (f) with a percentage to specify the amount of cases that should be affected by noise. The type of noise applied to each instance was selected randomly. In these experiments, we observed that it was possible to raise the amount of noise to as much as 40% (meaning that about 40 out of 100 instances were affected by noise) and still discover the correct cause for the delay. Above 40%, however, the decision tree stops capturing the correct cause for the delay.

VIII. Conclusion

In this paper we addressed the analysis of process delays from a classification perspective and proposed an ILP-based approach to discover the reason why some process instances become delayed. This approach relies on first-order logic for data representation, whose expressiveness can fully capture the knowledge embedded in an event log. The event log is expressed as a set of logic facts (knowledge base) and through a set of common rules (background knowledge) it is possible to derive additional facts. All of these facts may contribute to identify the cause for delay. This cause is expressed as a rule extracted from a logical decision tree that classifies all process instances according to duration.

We conducted several experiments with synthetic event logs, where we inserted deliberate delays in process instances according to a certain condition. The decision tree was able to capture that condition, although in some experiments it was necessary to include additional predicates to better express the cause for the delay. We also carried out experiments to study the scalability and noise-tolerance of the proposed approach.

The same approach can also take into account information about the data-flow perspective of the business process, if available, and we are planning to demonstrate that in future work. It can also be used to study other response variables besides duration. Currently, the use of predicates such as `handover/3` and `together/3` provides a way to explain the occurrence of delays based on the interaction or participation of certain users, and this provides an indication of the effectiveness of such collaborations in the process.

References

[1] W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 1128–1142, 2004.

[2] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. de Medeiros, "Process mining with the HeuristicsMiner algorithm," BETA Working Paper Series, Eindhoven University of Technology, Tech. Rep. WP 166, 2006.

[3] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Genetic process mining: an experimental evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[4] W. M. P. van der Aalst and M. Song, "Mining social networks: Uncovering interaction patterns in business processes," in *Business Process Management*, ser. LNCS, vol. 3080. Springer, 2004, pp. 244–260.

[5] W. M. P. van der Aalst, H. A. Reijers, and M. Song, "Discovering social networks from event logs," *Computer Supported Cooperative Work*, vol. 14, no. 6, pp. 549–593, 2005.

[6] P. Hornix, "Performance analysis of business processes through process mining," Master's thesis, Eindhoven University of Technology, 2007.

[7] R. S. Mans, "Workflow support for the healthcare domain," Ph.D. dissertation, Eindhoven University of Technology, 2011.

[8] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. van der Aalst, and P. J. M. Bakker, "Process mining in healthcare – a case study," in *Proceedings of the First International Conference on Health Informatics (HEALTHINF 2008)*, vol. 1. INSTICC, 2008, pp. 118–125.

[9] S. Suriadi, C. Ouyang, W. M. P. van der Aalst, and A. ter Hofstede, "Root cause analysis with enriched process logs," in *Business Process Management Workshops*, ser. LNBIP, vol. 132. Springer, 2013, pp. 174–186.

[10] A. Rozinat and W. M. P. van der Aalst, "Decision mining in ProM," in *Business Process Management*, ser. LNCS, vol. 4102. Springer, 2006, pp. 420–425.

[11] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan, "Business process intelligence," *Computers in Industry*, vol. 53, no. 3, pp. 321–343, April 2004.

[12] N. Lavrac and S. Dzeroski, *Inductive Logic Programming: Techniques and Applications*. New York: Ellis Horwood, 1994.

[13] H. Blockeel and L. D. Raedt, "Top-down induction of first order logical decision trees," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 285–297, May 1998.

[14] S. Goedertier, D. Martens, B. Baesens, R. Haesen, and J. Vanthienen, "Process mining as first-order classification learning on logs with negative events," in *Business Process Management Workshops*, ser. LNCS, vol. 4928. Springer, 2008, pp. 42–53.

[15] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari, "Exploiting inductive logic programming techniques for declarative process mining," in *Transactions on Petri Nets and Other Models of Concurrency II*, ser. LNCS, vol. 5460. Springer, 2009, pp. 278–295.

[16] L. D. Raedt and W. V. Laer, "Inductive constraint logic," in *Algorithmic Learning Theory*, ser. LNCS, vol. 997. Springer, 1995, pp. 80–94.

[17] H. M. Ferreira and D. R. Ferreira, "An integrated life cycle for workflow management based on learning and planning," *International Journal of Cooperative Information Systems (IJCIS)*, vol. 15, no. 4, pp. 485–505, 2006.

[18] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[19] ——, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[20] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Chapman and Hall, 1984.

[21] H. Blockeel, "Top-down induction of first order logical decision trees," Ph.D. dissertation, Department of Computer Science, K.U. Leuven, Belgium, December 1998.

[22] L. Maruster, "A machine learning approach to understand business processes," Ph.D. dissertation, Technische Universiteit Eindhoven, The Netherlands, 2003.

[23] A. K. A. de Medeiros, "Genetic process mining," Ph.D. dissertation, Technische Universiteit Eindhoven, The Netherlands, 2006.

[24] S. Goedertier, "Declarative techniques for modeling and mining business processes," Ph.D. dissertation, K.U. Leuven, Belgium, 2008.